Е. М. Патрикеев

Автоматический Синтез Параллельнных Программ

Научный руководитель: д.ф.-м.н. В. Б. Новосельцев

Аннотация. Автоматический синтез программ является одним из разделов искусственного интеллекта и нацелен на генерацию программ, в соответствии с заданными условиями и имеющимися „знаниями“ (информацией о решении различных задач). В связи с развитием многопроцессорных ЭВМ, и полномасштабной интеграцией параллелизма в разрабатываемые сегодня программы и алгоритмы, автоматическая генерация параллельных программ также становится актуальной. В работе описывается метод поиска точек распараллеливания в программе, синтезируемой методом, предложенным Новосельцевым В.Б и Пинжиным А.Е.

1. Введение

Проблема Искусственного интеллекта имеет достаточно богатую историю и является предметом обсуждений и споров для философов уже многие века. Однако как научное направление, ИИ берет своё начало только в середине XX века, сразу за развитием теории алгоритмов и созданием ЭВМ. С прогрессом в области информационных технологий, компьютеры переходили из вычислительных машин, выполняющих строго определенные последовательности команд, в интеллектуальные помощники, способные самостоятельно решать некоторый круг задач и строить программы, основываясь на данных о предметной области и накопленном опыте. Для успешного построения и синтеза таких программ необходимо смоделировать интеллектуальную деятельность человека и формализовать такие интеллектуальные понятия, как „знание“ или „вывод“. Здесь модель может различаться из-за различий в подходе человека к решению задачи и процесса его мышления. Так, существуют *индуктивный* ([8] и [9]), *трансформационный* ([10] и [11]) и *дедуктивный* ([2] и [3]) подходы к автоматическому синтезу программ.

В данной работе рассматривается и модернизируется метод [1], основанный именно на дедуктивном подходе, основной задачей которого является построение интуиционистского доказательства теоремы существования решения, из которой и получается программа, удовлетворяющая заданным условиям (из такой постановки также исходят [5] и [12]).

В связи с широким распространением мультипроцессорных компьютеров, возникает необходимость дополнения алгоритма синтеза процедурой поиска „точек распараллеливания“ или процедур, которые могут быть выполнены параллельно - абстрагируясь, для начала, от необходимости непосредственного распределения процедур по процессорам.

Благодаря особенностям представления знаний в теории функциональных моделей (см. главу 2), может быть реализовано *крупноблочное* распараллеливание - параллельные части программ (блоки) могут быть выделены на основе объектов этой теории и точки распараллеливания могут быть выявлены уже в процессе логического вывода.

2. Теория структурных функциональных моделей

Теория структурных функциональных моделей предложена В.Б. Новосельцевым и основана на моделях, введенных Э.Х.Тыугу [6] и А.Я.Диковским [4] и полном исчислении [7], не выводящем за рамки исчисления высказываний. Теория обладает необходимой и достаточной мощностью для создания строгих и в то же время выразительных моделей „знаний“ (исходных данных), естественным образом проецируется на термины и конструкции структурного программирования [5] и весьма эффективна при описании методов планирования и синтеза. Введем основные понятия этой теории:

Определение 1.1. Описание модели предметной области Σ, называемое сигнатурой, состоит из четырех не более чем счетных множеств (A, F, P, D) и представляет собой множество имен, функциональных символов, селекторных символов и имен схем отношений соответственно. Считается также, что выделено некоторое непустое подмножество имен первичных или примитивных схем $E \in D$.

Определение 1.2. Атрибуты, составляющие множество A, представляют собой именованные типизированные объекты предметной области. Каждый атрибут связан со схемой из множества D. Если

T∈D\E (то есть схема не примитивна), то связь атрибута a со схемой T выражается записью T(a) и атрибут называется непервичным. В противном случае, если T∈E, то атрибут является примитивным (например, целое число int, строка string) и запись T(a) может быть заменена просто на a, если ссылка на схему не имеет значения.

Определение 1.3. Выражение вида:

$$f : a_1, \ldots, a_n \to a_0, \tag{1}$$

где a_i, i = 0...n - имена атрибутов, называется функциональной связью (далее - ФС), и обозначает возможность вычисления атрибута-результата a_0 по значениям атрибутов-аргументов $a_1, \ldots, a_n$ с помощью отображения f. При описании модели предметной области на языке структурных функциональных моделей важнейшим является понятие отношения над атрибутами или схемы.

Определение 1.4. Схема (отношения) T атрибута t определяется выражением вида:

$$\begin{array}{l} \mathrm{T(t)} = \mathrm{(t).}(S_{01}(a_{01}), \ldots, S_{0N0}(a_{0N0}) \mid filter_0 \\ \quad \text{if} \quad p_1 \Rightarrow S_{11}(a_{11}), \ldots, S_{1N1}(a_{1N1}) \mid filter_1 \ \square \ldots \square \\ \qquad p_k \Rightarrow S_{k1}(a_{k1}), \ldots, S_{kNk}(a_{kNk}) \mid filter_k \ \ \text{fi}), \end{array}$$

где t∈A - имя атрибута схемы, T∈D\E обозначает имя схемы. Для всех возможных значений индексов i, j символы S_{ij}∈D обозначают собственные подсхемы, a_{ij}∈A - собственные атрибуты схемы T, а $filter_0$ обозначает список собственных ФС. Часть выражения, находящаяся между обозначениями if... fi является опциональной и определяет вариантную или условную часть схемы. Вариантная часть состоит из условных ветвей, разделенных символом □, которые соответствуют традиционной программной конструкции if... elseif... else или ветвям оператора case. В условных ветвях символы p_i∈P, i=1...k называются выбирающими селекторами и представляют собой логическую функцию от собственных атрибутов.

Определение 1.5. Конечная совокупность схем M=$(T_1, \ldots, T_s)$ называется структурной (C-) функциональной моделью. Для пояснения введенных обозначений приведем пример модели, описывающей простейшую систему уравнений:

$$y = \left\{ \begin{array}{rcl} x & , & x < 0 \\ x + n & , & x \geq 0 \end{array} \right. , \ n = x^2.$$

Основная схема:

$$\begin{array}{l} \mathrm{T} = (\mathrm{x},\ \mathrm{y} \\ \quad \text{if } p_1(x) \supset \mid f_{11} : x \to y \ \square \\ p_2(x) \supset r : R \mid f_{21} : x \to r.m; f_{22} : r.n, x \to y \text{ endif }) \end{array}$$

Во второй ветви условия имеется атрибут r схемы R для вычисления квадрата числа: R = (m, n | $f_{sq} : m \to n$).

При этом, f_{21} имеет своим результатом атрибут m подсхемы R (вход в подсхему), а f_{22} имеет в аргументах атрибут n (выход из подсхемы). Далее возможно поставить задачу на модели (T,R).

Определение 1.6. Задачей планирования (вывода) на С-модели M называется тройка S=(A_0, X_0,T), где A_0 и X_0 - наборы имён соответственно исходных и искомых атрибутов, а T∈M - схема, в которой определены эти имена. Продолжая пример, может быть сформулирована следующая постановка задачи: S = (x, y, T).

Содержательно, задача планирования состоит в построении алгоритма, реализующего вычисление искомых атрибутов из исходных при помощи той или иной последовательности ФС. Схема алгоритма извлекается из доказательства соответствующей логической теоремы, в соответствии с интерпретацией (семантикой) модели, а само доказательство выводится в соответствующем исчислении SR [3] по определенным правилам вывода, составляя список объектов „шаг доказательства“. *Шаг доказательства* может быть, например, функциональной связью, ветвлением или вызовом подпрограммы.

Определение 1.7. Интерпретация I С-модели M задает:

- для каждой элементарной схемы S∈E непустое множество (первичный тип) $S|_I$;
- для каждого функционального символа f∈F - отображение $f|_I : S_1|_I \times \ldots \times S_n|_I \to S_0|_I$;
- для каждого селектора p∈P - булевское отображение $p|_I : S_1|_I \times \ldots \times S_n|_I \to \{\text{И, Л}\}$.

Таким образом, из схемы алгоритма при помощи интерпретации получается логическая программа, причем элементарные схемы переходят в типы, ФС в функции/операторы, селекторы в условия ветвлений. Так, в упомянутом примере $p_1(x)$ сопоставлен выражению $(x < 0)$, $p_2(x)$ выражению $(x \geq 0)$, f_{11} и f_{12} - оператору присваивания, а f_{22} - оператору сложения.

3. Алгоритм поиска точек распараллеливания

В данном алгоритме используется специфика стратегии вывода [1], позволяющая в некоторый момент „рассудить“, смогут ли выводимые процедуры быть выполнены параллельно в будущей программе.

При постановке задачи (A_0, X_0, S), вывод происходит по следующей схеме:

```
«создать список достижимых атрибутов C_d = A_0»;
«поднять флаг ‘‘продолжать доказательство’’»;
«текущей схемой объявить исходную схему S»;
while «поднят флаг ‘‘продолжать доказательство’’» do
   if «имеется ФС текущей схемы, аргументы которой входят в C_d» then
      «применить одно из правил вывода»;
      «добавить к C_d новые достижимые атрибуты»;
      «если определился вход в подсхему, добавить её в стек подсхем»;
   else if «имеется подсхема в стеке, в которую определён вход» then
      «убрать её из стека и объявить текущей схемой»;
   else if «текущая схема не исходная» then
      «объявить текущей внешнюю схему»;
   else «опустить флаг ‘‘продолжать доказательство’’»; od
if «C_d ⊃ X_0» then «задача решена»;
```

Вкратце, алгоритм выводит всё, что возможно на текущей схеме, и только после этого „спускается“ в одну из подсхем по входу в подсхему. При этом, если определен вход в несколько подсхем, вывод на них может происходить независимо (поскольку он происходит обособленно, внутри каждой подсхемы), а следовательно и интерпретации ФС разных подсхем (суть функции) смогут выполняться независимо, т.е. параллельно. Соответственно, если при переходе от доказательства к логической программе каждую последовательность „шагов доказательства“, выведенную на отдельной схеме, выносить в отдельную процедуру (как сделано в [1]), эти процедуры можно будет выполнить параллельно.

Дополнение в алгоритм вывода, производящее поиск точек распараллеливания, выглядит следующим образом:

```
...
else if «имеется подсхема в стеке, в которую определён вход» then
  if «из текущей схемы определено несколько входов в подсхемы» then
     «пометить процедуры, выведенные на этих подсхемах
```

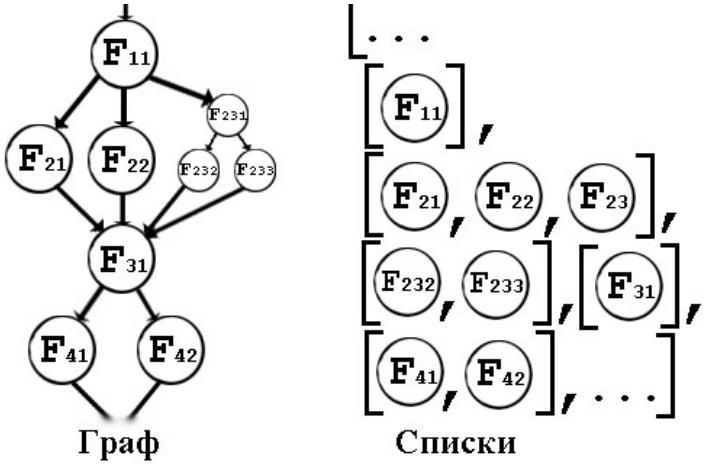

Рис. 1. Примеры представления параллельных процедур

```
                как выполняемые параллельно»;
  «убрать подсхему из стека и объявить текущей схемой»;
else if «текущая схема не исходная» then
...
```

Несмотря на краткость дополнений в псевдокоде, при реализации необходимо будет учесть некоторые моменты:

1) когда имеет место условие *вход из текущей схемы определен более чем в одну подсхему*, вывод на схемах еще не закончен и процедуры не сформированы - необходимо помечать схемы вместо процедур и в дальнейшем при необходимости заменять их, либо создавать взаимо-однозначное соответствие между схемами и процедурами;

2) представление параллельности тоже необходимо выбрать, к примеру: ориентированный граф с вершинами-процедурами или список списков процедур (см. рис 1);

3) необходимо избежать множественного внесения факта параллельности одних и тех же процедур, в зависимости от представления (п.2);

4) предусмотрена параллельность между вложенными процедурами, т.е. процедурами, вызванными в процедуре (учтено на рис. 1, см. F_{232} и F_{233}).

Таблица 1. Примерное описание процесса вывода

№	Достижимые атр-ы	Стек	Шаги доказательства
0	инициализация: исходная схема T - текущая схема		
1	[x, y(при условии p_1)]	[]	f_{11} при условии p_1 (вариантная часть)
2	[x, y/p_1, sq.m]	[SQ]	f_{11}/p_1; f_{21}/p_2 (вариантная часть)
3	[x, y/p_1, sq.m, cb.m]	[SQ,CB]	f_{11}/p_1; $(f_{21}; f_{22})/p_2$
4	помечаем подсхемы SQ и CB как параллельные спуск в подсхему		
5	[x, y/p_1, sq.m, cb.m]	[CB]	f_{11}/p_1; $(f_{21}; f_{22}; \{f_{sq}\})/p_2$
6	переход в подсхему CB		
7	[x, y/p_1, sq.m, cb.m]	[]	f_{11}/p_1; $(f_{21}; f_{22}; \{f_{sq}\}; \{f_{cb}\})/p_2$
8	подъем в исходную схему T		
9	[x, y, sq.m, cb.m]	[]	f_{11}/p_1; $(f_{21}; f_{22}; \{f_{sq}\}; \{f_{cb}\}; f_{23})/p_2$

Чтобы привести пример работы алгоритма, расширим предыдущую модель до двух подсхем. Для этого дополним систему:

$$y = \left\{ \begin{array}{rcl} x & , & x < 0 \\ sq + cb & , & x \geq 0 \end{array} \right. , \; sq = x^2, cb = x^3.$$

Основная схема: T = (x, y

$$\begin{array}{c} \text{if } p_1(x) \supset \mid f_{11} : x \to y \; \square \\ p_2(x) \supset sq : SQ, cb : CB \mid f_{21} : x \to sq.m; \\ f_{22} : x \to cb.m; f_{23} : sq.n, cb.n \to y \text{ endif }) \end{array}$$

Подсхемы:

$$\text{SQ} = (\text{m, n} \mid f_{sq} : m \to n), \; \text{CB} = (\text{m, n} \mid f_{cb} : m \to n)$$

В таблице 1 показан ход алгоритма. Из-за селекторов в схеме T, первым шагом доказательства становится вариантная часть, в которую входят условные шаги доказательства, первый из которых, ФС f_{11}, делает атрибут-результат y условно-достижимым. Поскольку необходима полная достижимость результатов, вывод продолжается.

После вывода на схеме T, обнаруживается и фиксируется параллельность между схемами SQ и CB и осуществляется вход в обе схемы по очереди и вывод на них - результатом являются два шага доказательства „вызов подпрограммы“, которые также содержат ШД, заключенные в фигурные скобки. Затем при помощи f_{23} устанавливается достижимость y при p_1 и p_2, что всегда истинно, а это значит полную достижимость.

По такому выводу получается следующий код:	или, если заменить ФС на функции и преобразовать if :

if (x<0) f11(x,y);
if (x>=0) {
SQ sq;
CB cb;
f21(x,sq.m);
f22(x,cb.m);
\\ parallel section
sq.m=Fsq(sq.m);
cb.m=Fcb(cb.m);
\\ end of parallel section
f23(sq.n,cb.n,y);
}
	if (x<0) y=x;
	else {
	SQ sq;
	CB cb;
	sq.m=x;
	cb.m=x;
	\\ parallel section
	sq.Fsq();
	cb.Fcb();
	\\ end of parallel section
	y=sq.n+cb.n;
	}

Из-за различных реализаций параллельности в программе, в данном примере „параллельная секция“ была отмечеными обычными комментариями.

Также, лучшей оптимизацией для данной задачи было бы изначальное вынесение квадрата $sq + cb{=}x^2 * (1 + x)$, однако это (наряду с некоторыми неточностями) было сделано для наглядности примера синтеза программы из постановки задачи и для акцента на том, что оптимизация должна проводиться различными методами, к различным частям программы на различных фазах её формирования.

4. Заключение

Как теоретическая наука, параллельные вычисления все еще являются предметом активного исследования и включают множество нерешенных задач. Так, для абстрактной программы в общем случае пока не существует эффективного распараллеливающего алгоритма

и параллелизация проводится скорее вручную (для каждой конкретной программы) а не автоматически.

Данный алгоритм также находится лишь в стадии разработки и будет расширяться (в том числе на циклы и рекурсии) по мере разработки собственно синтезатора [1] и интеллектуализации CASE[1]-систем.

Список литературы

[1] Пинжин А. Е. Алгоритмы интеллектуальной обработки информации и структурного синтеза программ. — Томский политехнический университет, 2009. — 110 с.

[2] Новосельцев В.Б. Синтез рекурсивных программ в системах управления пакетами прикладных программ. — Институт Теоретической Астрономии АН СССР, Ленинград, 1985. — 51 с.

[3] Новосельцев В.Б. Формальная теория структурных моделей описания информационных систем и методы установления выводимости. — Томский политехнический университет, 2006. — 207 с.

[4] Диковский А.Я. Детерминированные вычислительные модели // Техническая кибернетика, 1984. — 84-105 с.

[5] Непейвода Н.Н. Соотношение между правилами естественного вывода и операторами алгоритмических языков высокого уровня // Докл. АН СССР, 1978. — 526-529 с.

[6] Тыугу Э.Х. Решение задач на вычислительных моделях // ЖВМ и МФ, 1970. — 716-733 с.

[7] Минц Г. Е. Т. Э. Х. Полнота правил структурного синтеза // Докл. АН СССР, 1982.

[8] Барздинь Я.М. Об индуктивных правилах вывода для синтеза программ // Синтез, тестирование, верификация и отладка программ. Тезисы докл. всесоюзной научной конференции. — Рига, 1981. — 25-26 с.

[9] Барздинь Я.М. Один подход к проблеме индуктивного вывода // Применение методов математической логики. Тезисы докл. всесоюзной научной конференции. — Таллин, 1983. — 16-28 с.

[10] Burstall R. M. D. J. A system which automatically improves programs // Acta Informatica, vol. 6., 1976. — 41 - 60 p.

[11] Burstall R. M. D. J. A Transformation system for developing Recursive Programs // Journal of the ACM, vol. 24., 1977. — 44 - 67 p.

[12] Waldinger R.J. L. R. C. T. A step toward automatic program writing. // Proc. of the 1st International Joint Conference on Artificial Intelligence. — Bedford, 1969. — 241-253 p.

УГП, 5М51

[1]Computer Aided Software Engineering

E. M. Patrikeev. *Automatic Synthesis of Parallel Programs* // Proceedings of Junior research and development conference of Ailamazyan Pereslavl university. — Pereslavl, 2010. — p. 133–142. (*in Russian*).

Abstract. Automatic synthesis of programs is one of the areas of Artificial intelligence, aimed at program generating in concordance with given conditions (specification) and available "'knowledge"' (information about solutions to different problems). Because of the developement of multiprocessor computers and wide intergration of parallel technologies into programs and algorythms designed today, automatic parallel programs generating also becomes urgent. A method of finding "'parallel points"' in a program, synthesized by the method, presented by Novoseltsev V.B. and Pinzhin A.E., is described in this work.

Key Words and Phrases: